Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Diploma Thesis

Profiling and Detection of IoT Attacks in Telnet Traffic

Bc. Simona Musilová

Supervisor: Ing. Sebastián García, Ph.D.

Study program: Open Informatics
Specialisation: Cyber Security

January 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Musilová Simona** Personal ID number: **393083**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science and Engineering**

Study program: **Open Informatics**

Branch of study: **Cyber Security**

II. Master's thesis details

Master's thesis title in English:

Profiling and Detection of IoT Attacks in Telnet Traffic

Master's thesis title in Czech:

Profilování a detekce IoT útoků v telnetovém provozu

Guidelines:

In the last five year the prevalence of IoT devices opened the door to a myriad of different attacks on unprotected home devices. These devices came from the factory with several vulnerabilities that can not be fixed without replacing the device. The most used protocol for this IoT devices is the Telnet protocol. However, there does not exist any tool or research or methodology to protect the devices by studying the Telnet protocol.
As part of the research task, the thesis will figure it out how to analyse the telnet protocol in order to better protect the devices by profiling the behavior of the connections in the network and by building models of the users and attackers (including automatic attackers) in order to find the best way to stop the attacks by developing methodologies that rely on behavioral
techniques.
The analysis of the Telnet protocol, together with the new methodologies should help improve the detection of the attacks received from the external networks and the internal networks, including rogue users and bots.

Bibliography / sources:

Prokofiev, A. O., Smirnova, Y. S., & Silnov, D. S. (2017). The Internet of Things cybersecurity examination. Proceedings - 2017 Siberian Symposium on Data Science and Engineering, SSDSE 2017, 44–48. https://doi.org/10.1109/SSDSE.2017.8071962
Mcdermott, C., Mcdermott, C. D., Petrovski, A. V, & Majdani, F. (2018). Towards ituational Awareness of Botnet Activity in the Internet of Things. (August). Retrieved from https://www.researchgate.net/publication/326070511
Antonakakis, M., April, T., Bailey, M., Bernhard, M., Bursztein, E., Cochran, J., … Zhou, Y. (2016). Understanding the Mirai Botnet.
Metongnon, L., & Sadre, R. (2018). Beyond Telnet: Prevalence of IoT Protocols in Telescope and Honeypot. Proceedings of the 2018 Workshop on Traffic Measurements for
Cybersecurity - WTMC '18, 21–26. https://doi.org/10.1145/3229598.3229604

Name and workplace of master's thesis supervisor:

Ing. Sebastián García, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **02.07.2019** Deadline for master's thesis submission: _____

Assignment valid until: **19.02.2021**

Ing. Sebastián García, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce her thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Declaration

I declare that I elaborated this thesis on my own and that I mentioned all the information sources and literature that have been used in accordance with the Guideline for adhering to ethical principles in the course of elaborating an academic final thesis.

In Prague on January 03, 2020 .....

Acknowledgements

I would like to thank my supervisor Sebastian Garcia for regularly pushing me outside of my comfort zone. I would also like to express my deepest appreciation to all the teachers and classmates for sharing their knowledge and selfless help. I cannot leave CTU without mentioning my colleagues Anna and Michal, who provided me with unwavering support while writing this thesis.

Furthermore, I am extremely grateful to my partner Jan for his never-ending support through-out my studies, as well as our four-legged fluffy princesses for listening to my code. The completion of this thesis would not have been possible without the support of my best friends Mery, Vanda and Adam.

Last but not least, I also wish to thank my family and friends for their support and profound belief in my abilities even though I am sure none of them had any idea what I was usually talking about.

Thank you all for joining me on this extraordinary journey!

Abstract

The Internet of Things devices are getting more popular in everyday life. They can be found not only in smart homes but also in a variety of industries, from factories to health care. Many of those devices are using the Telnet protocol, which doesn't provide any encryption of the sent data. If the user's login credentials got compromised, there is no protection for the user. In the first part of this thesis, we created a method that can read and understand the Telnet traffic to provide an insight into all the connections coming to the Telnet server. In the second part of this thesis, we propose a method to create a new profile for every user connecting to the Telnet server. Based on comparing the key features of the client's behaviour, including the network-based keystroke dynamics, the proposed method can detect if a new connection is coming from a different user than the connections before. Based on this comparison, we can provide a new layer of security for every Telnet device connected to the Internet.

Abstrakt

IoT („*Internet of Things*“, internet věcí) se v běžném životě stává stále více populární. Najdeme jej nejenom v chytrých domácnostech, ale i v dalších odvětvích — od průmyslu po zdravotnictví. Mnoho těchto přístrojů využívá Telnet protokol, který nepodporuje žádné šifrování zasílaných dat. Jsou-li přihlašovací údaje kompromitovány, neexistuje žádná dodatečná uživatelská ochrana. V první části této práce jsme vytvořili metodu, pomocí které je možné číst telnetový provoz, porozumět mu a poskytnout náhled na všechna spojení, která přichází na telnetový server. V druhé části této práce navrhujeme metodu, jež vytvoří nový profil pro každého uživatele, který se připojí k telnetovému serveru. Na základě porovnání klíčových charakteristik chování klienta, včetně metody „*keystroke dynamics*“ v síti, dokáže navrhovaná metoda rozeznat, zda nové spojení přichází od jiného uživatele než předchozí spojení. Na základě tohoto porovnání můžeme poskytnout novou vrstvu ochrany pro každé zařízení připojené na internet a využívající Telnet.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Cybersecurity is a shared responsibility, and it boils down to this: in cybersecurity, the more systems we secure, the more secure we all are."

—Jeh Johnson
US Secretary of Homeland Security

The popularity of Internet of Things (IoT) devices has been rising dramatically in the previous years [9], reaching 26.66 billion IoT devices active on the Internet in 2019 [21]. IoT devices are getting more popular thanks to the range of services they can provide and their decreasing price. Every second, 127 new IoT devices are connected to the Internet worldwide [41].

When people think about IoT devices, they usually imagine a home equipped with smart voice assistants, light bulbs, door locks, coffee machines or other smart home devices. However, there are many more areas where IoT is used. New cars manufactured nowadays are connected to the Internet to receive updates of all the systems they are using; industrial IoT devices are used in many factories; also towns and cities are getting connected to the Internet. For example, in Zlín, a town in southeastern Moravia in the Czech Republic, the city mayor installed trash bins that can send a text message when they are full [37]. The health industry was also impacted by IoT devices having, for example, a patient's pacemaker connected to the Internet all the time.

Despite being ubiquitous, IoT devices suffer from many security problems. Maybe the most important of which is that most IoT devices are shipped with the same default credentials from the manufacturer. After delivering to the customer without enough knowledge of networking and security, the default credentials are never changed. Therefore, all those IoT devices are exposed to attacks on the Internet without almost any protection. This vulnerability is more important if we consider that the hardware of IoT devices can not be updated, and that firmware upgrades are difficult to do.

One of the most important caveats of IoT is the use of simple and old protocols for communication. One such system is Telnet, the remote terminal program, which is being

used for the simplicity of its implementation. Many IoT devices use the Telnet protocol with default login credentials for accessing its system. This is a huge problem and the source of many of the attacks seen in the IoT world. In an attempt to know how many Internet-facing Telnet servers were opened to the Internet we conducted small monitoring for 16 months (see Section 3.1), concluding that currently there are approximately 5 million devices with opened Telnet port connected to the Internet. The total amount of Telnet servers may be higher since many devices are located behind a firewall and therefore are not directly reachable from the Internet. However, these hidden devices can still be infected from another device in the internal network.

Despite being implemented since 1969 and being used in many devices, the Telnet protocol is not thoroughly processed and understood by most Intrusion Detection Systems and network analysers. It is not clear why most systems can not understand Telnet protocol, but it may be because the resurgence of Telnet was due to IoT devices, and the problem of IoT security is very recent. It may also be because most IDS focus on business networks that are usually correctly configured not to use the Telnet protocol. However, since most IoT devices are left unprotected and vulnerable to attacks, there is a need for a tool that can understand and process the Telnet protocol.

The need for a Telnet protocol analyser and security detector is large since so far there was no way to tell if there was any attempt to login to a Telnet server by automatically analysing the network traffic. No tool so far could extract used credentials from login attempts, and therefore any automated analysis of the Telnet sessions was impossible. The only known way to monitor attacks against Telnet was by controlling the Telnet device itself.

In addition to the lack of Telnet analysers, there was no protection for users who never changed the default credentials and for users whose credentials were compromised. There was no way to protect a Telnet server and to detect that an unauthorised person logged in to the Telnet server.

The aim of this thesis is to create a new layer of security for Telnet devices in the network that can (1) understand the Telnet protocol from the network packets, (2) profile the users and Telnet sessions, (3) detect when the logged-in user is not the owner of the account, (4) provide statistics about the connections and sessions. Our new approach can provide enough data for monitoring Telnet attacks and the behaviour of attackers without the need of controlling the Telnet devices themselves.

The first part of this thesis is to create a new analyser for the Telnet protocol from traffic captured in the network. The results should provide information about every IP address connecting to a Telnet server, including the distribution of used credentials. It is necessary to open every captured packet, understand the TCP protocol, the Telnet protocol, and then extract all the information to provide statistics about the sessions.

The second part of this thesis aims to profile Telnet users based on their behaviour. The features are based on the regular time and place of connection to the device and all the commands used in the communication. Apart from the sent data, the proposed method also captures how the user is typing on the keyboard, implementing a network-based keystroke dynamics method. The goal is to detect if the current typing user is the real logged-in user

in the device or not, in order to protect users in situations where their credentials were compromised.

The idea of our proposed method and its partial results were presented on two IT conferences. On 31 October 2019, the research was introduced on the OWASP Czech Chapter meeting [13] in Prague, Czech Republic. Later on 9 November 2019, the research was presented on DevFest 2019 [8] in Libčice nad Vltavou, Czech Republic. On both events, the study was presented with a title *When A Password Is Not Enough: Developing A New Way Of Protecting Smart Homes*. On 10 December 2019, the research was briefly introduced at the IoT Lab Hackaton organised by the Avast IoT Lab. The title of the talk at this event was *Detection of IoT Attacks in Telnet Traffic*.

The rest of this document is organised as follows: Chapter 2 provides a review of previous work in related problems solved in this thesis. This chapter focuses on 4 different problems: the security of the Telnet protocol, attack detection on IoT devices, user authentication based on keystroke dynamics techniques and user profiling in the network. Chapter 3 provides all the information that the reader should know about the Telnet protocol, starting with the usage of this protocol and the meaning of all bytes in the communication. Next, the methods of parsing, restoring and interpreting the commands are presented. Chapter 4 describes the methods of analysing the Telnet traffic, as well as the statistics provided by the proposed method from the fast analysis of client's IP addresses, to the more complex analysis of sessions and used credentials, to the behavioural analysis of the users. Chapter 5 presents results obtained by using this method on the Telnet traffic captured in 70 consecutive days. Lastly, Chapter 6 concludes this thesis and discuss future remarks.

Chapter 2

Related work

This Chapter provides a review of previous work done in all the topics included in this thesis. The first section focuses on the Telnet protocol, its security, and ways of protecting the data of the Telnet user are presented. The second Section introduces some approaches to the analysis of IoT attacks. Different methods use different input data captured at various places in the network. The next section provides a review of user authentication methods based on the characteristics of a user's typing. The input data for these methods are either coming from the host computer or from the captured network traffic. Finally, the last Section reviews profile-based methods to detect attacks on any device in the network.

2.1 The security of the Telnet protocol

Telnet, also called the mother of all application layer protocol, has been used on the Internet for more than 50 years [31]. Back in the time, the aim was to connect two computers together to allow bi-directional terminal communication between them. At that time computer science specialists didn't worry about encrypting the data sent over the Internet.

The first Internet standard defining the Telnet protocol was *RFC 854* and it is dated on May 1983 [42]. In this document all the specifications of the Telnet protocol were described along with the basic information about options that can be used in the communication. More details about the options are described in the following documents from the same year, specifically in *RFC 855* [42], *RFC 856* [42], *RFC 857* [42], *RFC 858* [42], *RFC 859* [42], *RFC 860* [42] and *RFC 861* [42]. None of these Internet standards provides any encryption of sent data or any other protection of user's credentials.

The first Internet standard about securing the communication between client and server came in January 1993 with the *RFC 1409* [23], shortly obsoleted by *RFC 1416* [24]. They came with a new Telnet option for user authentication without the need of typing and sending a password in clear text to the Telnet server. This methodology was based on using other technologies like Kerberos protocol for authenticating the server and the client or RSA algorithm for encrypting the payload. This Internet standard was obsoleted later in 2000

by *RFC 2941* [46] where more technologies for securing user's data were implemented, for example, the SSL protocol.

The security of the Telnet protocol was not researched only to create a new Internet standard. In 2003 Mahmood [38] came with a new method to secure data sent over the Internet. They developed a new prototype system called *SecTel* which works on the Transport layer of the TCP/IP model. This new security layer encrypts the Telnet communication on the Transport layer by using a *SecTel* service before the client logs in to the Telnet server. To use this system, it is necessary to install the *SecTel_ Client* service on the client's computer and the *SecTel_ Server* service on the Telnet server.

Later in time, the researchers focused on creating a new secure protocol instead of securing the Telnet protocol. In January 2006 a new encrypted protocol called Secure Shell (SSH) was introduced with the *RFC 4250* [36]. Therefore the Telnet protocol was being replaced by SSH on the computers and server. But as the IoT devices are getting more popular, the Telnet protocol is being redeployed and used again on the Internet. Many small IoT chips have only a small amount of resources, and therefore the Telnet protocol is being used again because of its simplicity [34].

2.2 Detection of IoT attacks

The research in the field of detection of IoT attacks and IoT malware has started relatively recently with the increasing number of IoT devices connected to the Internet. The preferred method to learn about the IoT attacks on the Internet is to set up honeypots and monitor all the traffic coming to them. Based on this approach, researchers can understand the attack vectors of different malware families and observe how the trends are changing in time [26].

In 2019 Vishwakarma et al. proposed a method to recognize which malware family is attacking an IoT device [48]. They set up honeypots to capture real attacks on the Internet to collect log files containing the characteristics of the attack. Based on the behaviour characteristics of the attack, they can recognize a (known or yet unknown) malware family spreading across the Internet. Their method also provides a machine learning-based framework that can categorize the malware in real-time.

Most of the available statistics about IoT malware come from the data collected on the honeypots. There is a lack of available options to detect IoT malware only from the captured network traffic. The well-known intrusion detection systems, for example Zeek (formerly Bro) [20], Suricata [19] or Snort [18] can detect traffic going to the Telnet port. Still, they don't provide any additional details about the attack itself.

In 2019 Kumar et al. proposed a new method to detect IoT attacks based on their network activity [33]. They studied the characteristics of the IoT malware families and implemented a method to categorize them by using machine learning techniques. Their research focused on the three categories: the malware attacking the Telnet port, the malware using the HTTP GET request and the malware using the HTTP POST request to infect an IoT device.

2.3 User authentication based on typing characteristics

The researchers soon realised that using only the username and password for user authentication may not be enough to protect any system. Many methods enhancing the security of such devices were proposed and implemented in real life. One of them is based on the behavioural characteristics of a user's typing, called the keystroke dynamics method [39]. These characteristics are based on the rhythm and pattern or user's typing, and they are unique for each individual.

Over time many researchers focused on developing methods using the keystroke dynamics to create a new layer of security for systems. These methods aim to determine if the user using the system now is the same individual as the real user of the system [39]. The decision is made based on comparing typing characteristics of the user connected now to the server and the previously stored characteristics of the real user.

Two users can be compared based on different extracted features. Douhou et al. created a comparing method using only two features: the duration that the key is pressed and the duration between keystrokes [27]. Other authors used more features to obtain better results. Jadhav et al. proposed a method that extracts more keystroke dynamics features, for example, duration between releasing a key and releasing the following key [29]. Vinayak in his work extracted also the number of times the user pressed *SHIFT*, *BACKSPACE* and *CAPS LOCK* keys [47].

The decision methods mentioned above are all based on statistical tests. They provide the probability of new values coming from the same distribution as the stored values [27][29][47]. Another approach using Support Vector Machines was introduced by Sawant et al.[43]. All these methods capture data from users using an application running on the users' computers, for example, a web page with a login form. Based on this fact, the researchers could collect detailed typing characteristics.

Song et al. analysed the possibility of understanding the data send using the SSH protocol [44]. If the SSH is in the interactive mode, every key typed by the user is sent encrypted to the server right after typing it. Based on their research, the keystroke dynamics characteristics are preserved when bytes are sent to the network. They captured the traffic in the network and proved that it is possible to understand the user data.

2.4 Profiling users in the network

The analysis of characteristics of the user's typing is an efficient way to detect any unauthorised access to the user's account. Although it is still hard to identify those attempts only by reading the network traffic. The research on this topic is relatively new.

A new method to detect abnormal behaviour of the user in the network traffic was introduced by Kasimov in his master's thesis [30]. His method focused on analysing the Argus flows [16] in given time windows. This approach is based on high-level behaviour analysis of users and detecting any unusual changes in the user's behaviour.

Kubeša introduced a new method focusing on identifying users based on their behaviour in the network [32]. This method is also using the Argus flows [16] as input data. A new profile is created for each user in the network traffic. The user profiles are not based only on the IP address of the device or the payload data, but on a dozen of features in total.

Another method based on detection behaviour anomalies was presented by Tahir et al. [45]. Their method is also using the Argus flows [16] as input data to create user profiles in time windows. Every user's profile is compared with the user's activity in the past. The proposed method is detecting any behavioural abnormalities in the user's behaviour.

Profiling users is getting known also in the commercial sphere. Cisco developed a tool called *FireSIGHT* [1] to detect any anomaly behaviour. The tool needs to be set up manually to create a user profile in a specified time window. This profile is later used to be compared with any other activity. The *FireSIGHT* can trigger alerts to a user to notify him about any anomalies, although the rules to trigger these alerts need to be set up manually beforehand.

Chapter 3

Introduction to Telnet protocol

The Telnet protocol is one of the oldest application layer protocols in the TCP/IP model. It was developed in 1969, starting with RFC 15 [25]. The Telnet protocol provides bi-directional communication between two terminals over the Internet, usually using port 23/TCP or 2323/TCP. All the payload sent over the Telnet protocol is sent in clear text without any encryption. This fact allows anyone to read all the user data easily, including the credentials.

3.1 The usage of Telnet

The Telnet protocol is more than 50 years old, but it is still being used nowadays, mostly in the cheap and unsecured Internet of Things devices. According to the McKinsey Global Institute, there are 127 new IoT devices connected to the Internet every second [41]. Some of those devices are left with default credentials and exposed to the Internet waiting to be attacked.

One of the very well-known botnets spreading over the Internet is the Mirai botnet. Mirai was first found in August 2016 [4] attacking the Telnet ports 23/TCP and 2323/TCP. To gain access to the attacked device, Mirai used a list of 68 default usernames and passwords [3]. The Mirai botnet is also responsible for a massive DDoS attack against the KrebsOnSecurity's servers in September 2016 [2].

In 2018 Kaspersky Lab published statistics about attacks against their honeypots. In the Q2 of 2018, more than 75% of all attacks against the Kaspersky Lab honeypots were on the Telnet port. On the SSH port, there were 11.6% of all attacks and the remaining 13% of the attacks were against all other ports [35]. The attack distribution can be seen in Figure 3.1.

The number of attacks and connection attempts is indicating that there is a lot of devices with Telnet port opened connected to the Internet. To get a better idea about the real usage of the Telnet protocol, we were capturing the number of devices with Telnet port 23/TCP or 2323/TCP port opened to the Internet. From 12 September 2018 to 25 December 2019,

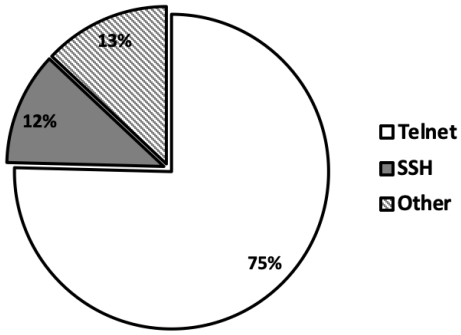
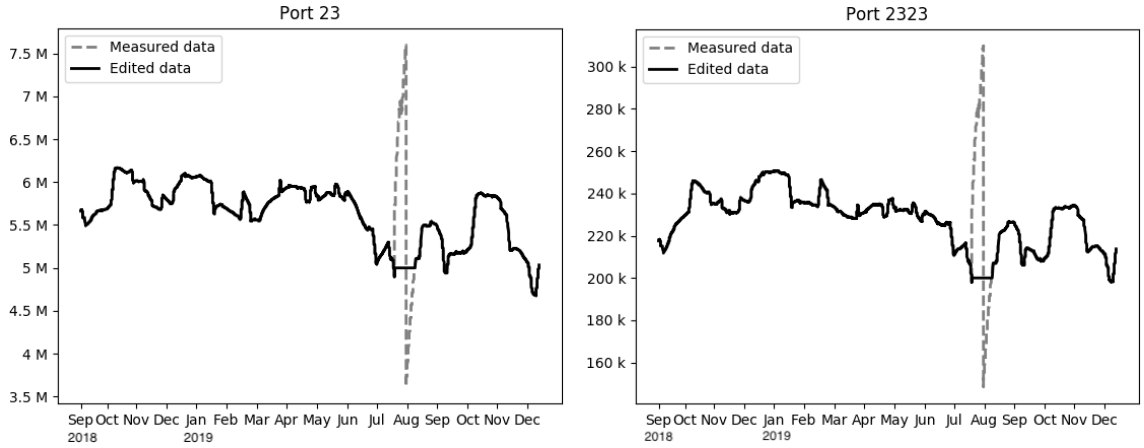


Figure 3.1: The number of attacks against honeypots in Q2/2018 [35]

we were capturing the number of devices with opened Telnet port from the Shodan search engine [17]. We requested the number of those devices every hour, and we stored the result.

The number of Telnet devices with port 23/TCP opened can be seen in Figure 3.2a, the number of devices with port 2323/TCP opened can be seen in Figure 3.2b. During the time we were capturing the data, we noticed a significant spike in the number of Telnet devices in summer 2019 on both ports, 23/TCP and 2323/TCP. From 30 July 2019, the number of Telnet devices was rapidly increasing, reaching its maximum on 11 August 2019. During the very same day, in only 6 hours the number of Telnet devices rapidly decreased to less than half of the previous value. From that moment, the number of Telnet devices was increasing again until it got back into a stable state. This spike is shown in Figure 3.2a and Figure 3.2b below in the grey dashed line. The number of captured devices can be seen in Table 3.1.



(a) Port 23 (b) Port2323

Figure 3.2: The number of devices with opened Telnet port on the Internet

Shortly after capturing this unexpected spike in the number of Telnet devices on the Internet, we contacted the Shodan representatives. We believed there must have been some problem with their service. They confirmed that the unusual values were caused by switching the search index clusters in their servers, so for a short time, the results could contain duplicate values. The time frame when the spike occurred matches this event.

Date	Port 23	Port 2323
30 July, 2019	5 million	200 000
11 August, 2019	7.6 million	300 000
11 August, 2019	3.6 million	148 000

Table 3.1: The number of Telnet devices in the Internet

3.2 Understanding Telnet bytes

Since the Telnet protocol is not encrypting the payload, it is easy to read all the communication between devices from the network traffic. All bytes in the Telnet protocol can be divided into two groups:

- **Negotiation bytes:** Dedicated non-printable bytes used to configure the two terminals and to exchange information about each other. Most of the negotiation bytes are transmitted at the beginning of the communication, but they can also be sent later at any time in the communication.
- **User data:** Every other byte in the Telnet communication that is not the negotiation byte. These bytes contain everything the user typed on the keyboard, and all the data user can see in their terminal.

3.2.1 Negotiation bytes

The negotiation process contains three bytes. The first byte is always `0xFF`, which indicates the start of the negotiation bytes. This byte is called *Interpret As Command (IAC)* byte.

The second byte is the option code indicating the desire to begin or stop performing a given option. The third byte is the option byte which specifies a function or setting of the terminal. In the negotiation process, four different bytes can be used to turn on or off an option [42]:

- `0xFE`: Demand to stop using specified option (DON'T)
- `0xFD`: Request to start using specified option (DO)
- `0xFC`: Reject the proposed option (WON'T)
- `0xFB`: Accept the proposed option (WILL)

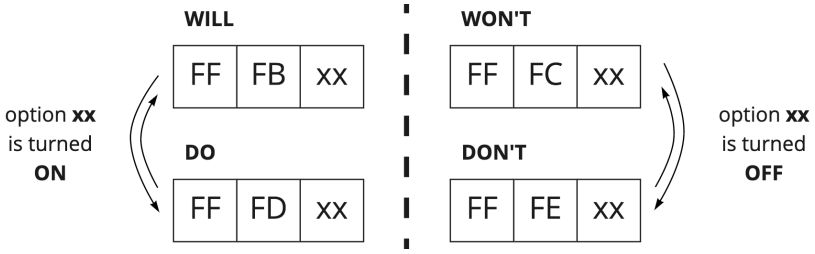


Figure 3.3: The structure of the Telnet negotiation bytes

The specified option is turned on only if both devices agree on using it. Any device can propose to use any option by sending the `0xFD` byte, also called the *DO* byte to the other device. If the other device wants to use this option, it sends back the `0xFB` byte, also called the *WILL* byte to accept the proposed option. If this device wants to reject the proposed option, it sends the `0xFC` byte, the *WON'T* byte. Any device can demand to turn off any previously agreed upon option by sending the `0xFE` byte, also called the *DON'T* byte to the client. This has to be confirmed by the other device that sends the `0xFC` byte, the *WON'T* byte to stop using the option. The negotiation bytes can be seen in Figure 3.3.

When the specified option is turned on, the subnegotiation process can start. In this process, the terminals exchange detailed information or settings about each other. The subnegotiation process always starts with the `0xFF` byte followed by the `0xFA` byte indicating the start of the subnegotiation bytes. The following byte is the specified option that was previously turned on, followed by any number of bytes. The meaning of those bytes depends on the specified option. The subnegotiation bytes always end with the `0xFF` byte followed by the `0xF0` byte. The negotiation bytes can be seen on Figure 3.4.

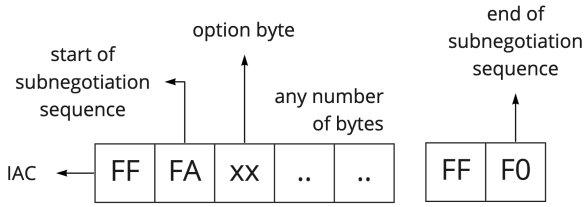


Figure 3.4: The structure of the Telnet subnegotiation bytes

The most of the negotiation and subnegotiation bytes can be found at the very beginning of the connection, between the TCP 3-way handshake and the login prompt from the Telnet server. Although some negotiation and subnegotiation bytes can appear at any moment during the session.

3.2.2 User data bytes

All the bytes that do not belong to the negotiation or subnegotiation byte structure are the user data bytes, which is the interaction between two hosts. Among those bytes in the server-client oriented connection, we can find everything the user on the client-side

was typing on the keyboard, editing the text and sending it to the server, as well as all the answers from the server.

In the server-client oriented communication, the user on the client-side is interacting with the server by typing commands in the command line. Every key that the user typed on the keyboard is sent to the server to process. Most of the sent bytes are returned back to the client and only after they are delivered to the client, they are displayed on the screen of the user's terminal window, see Figure 3.5. All those returned bytes are called the *echo bytes*. This is the reason why when a user is connected to a server far away from their location, the user can experience a delay between pressing a key and seeing the character on the terminal screen. There are some exceptions when the server is not sending back the echo bytes, for example, while typing a password or when sending special bytes.

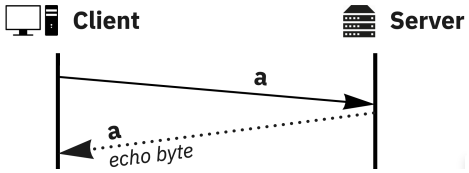


Figure 3.5: Telnet echo bytes

3.3 Parsing Telnet payload

There are a lot of Telnet libraries providing a function to create a new Telnet server or to connect to a Telnet server. But no library can open the Telnet communication, understand it, extract the credentials and commands from the network and extract other characteristics of the user. For this reason, we started with building our own method for parsing Telnet traffic and understanding what the client was doing.

As input data, we use traffic captured to the device in the middle of the two communicating devices. The analysis starts with reading and analysing every single packet coming from the network. If the source or destination port of a packet in not 23/TCP or 2323/TCP, the packet is ignored.

The Telnet protocol is an application layer protocol in the TCP/IP model, see Table 3.2. To find out what was sent in the payload, it is necessary to understand how the TCP protocol on the transport layer works. Only after that, the analysis of the Telnet protocol is possible.

3.3.1 TCP protocol

The main difference between the TCP and UDP protocol is that the TCP protocol guarantees the delivery and the order of packets sent to the Internet. This is done by using a sequence and acknowledge numbers in the TCP header of packets.

The sequence number is randomly generated at the beginning of each connection. Both client and server generate their own sequence number during the TCP 3-way handshake.

Layer	**Protocol**
Application	Telnet
Transport	TCP
Internet	IP
Network Access	Ethernet

Table 3.2: The TCP/IP model of the Internet

With each new packet sent, the sequence number increases by the length of the payload sent in the packet. In other words, the sequence number is not unique only for the packets sent to the network, it is unique and increasing with every new byte sent to the other host. This means that based on the sequence number, it is possible to put all captured bytes in the correct order.

The acknowledge number is sent by the receiving device back to the originator. It is set to the largest sequence number that was received without any missing byte before it. The acknowledge number is telling the originator if the previously sent bytes were delivered or not. If some bytes were not delivered to the receiving device, the originator sends them again with the same sequence number. This process is called the retransmission of bytes.

Ordering the bytes based on the sequence number solves some of the TCP issues. When a packet gets lost in the network, it is sent again by the source device. Since the input data for this method are packets sniffed on one place on the Internet, there may be duplicated bytes or packets in the capture. By ordering all bytes by the sequence number the duplicated bytes can be detected and ignored.

The method proposed in this thesis extracts the payload bytes and the sequence number of each byte from captured packets. The list of sent bytes is kept separate for the client and the server. For each byte, the time of capturing the byte is also stored for extracting user's typing characteristics later.

After analysing the TCP protocol values, the method creates a byte stream of all bytes sent to the Internet by a device. For each device, there is a separate stream of sent bytes. The beginning of communication coming from a client to the server can be seen in Figure 3.6.

```
ff fb 25 ff fd 03 ff fb 18 ff fb 1f ff fb 20 ff fb 21 ff fb 22 ff fb 27 ff
fd 05 ff fc 23 ff fa 1f 00 c5 00 34 ff f0 ff fa 20 00 33 38 34 30 30 2c 33
38 34 30 30 ff f0 ff fa 27 00 00 55 53 45 52 01 73 69 6d 69 ff f0 ff fa 18
00 58 54 45 52 4d 2d 32 35 36 43 4f 4c 4f 52 ff f0 ff fc 01 ff fd 01 70 69
72 61 74 65 0d 00 68 79 70 72 69 6f 74 0d 00 6c 73 0d 00 6d 6b 64 69 72 20
74 65 6c 6e 65 74 74 0d 00 63 64 20 74 65 6c 09 74 0d 00 6c 6c 7f 73 0d 00
76 69 20 6c 6f 67 2d 66 69 6c 65 2e 6c 6f 67 0d 00 1b 5a 5a 5b 41 ...
```

Figure 3.6: The stream of all bytes in the Telnet communication

3.3.2 Telnet protocol

As mentioned in Section 3.2, there are two types of Telnet bytes: the negotiation bytes and the user data bytes. The negotiation bytes are easy to parse since they are in a given format, see *RFC 854* [42]. However, those bytes don't characterise the user at all so they can be ignored, see Figure 3.7. Most of the Telnet negotiation bytes occur at the very beginning of the connection. But in general, negotiation bytes can be sent at any time during the communication.

```
ff fb 25 ff fd 03 ff fb 18 ff fb 1f ff fb 20 ff fb 21 ff fb 22 ff fb 27 ff
fd 05 ff fc 23 ff fa 1f 00 c5 00 34 ff f0 ff fa 20 00 33 38 34 30 30 2c 33
38 34 30 30 ff f0 ff fa 27 00 00 55 53 45 52 01 73 69 6d 69 ff f0 ff fa 18
00 58 54 45 52 4d 2d 32 35 36 43 4f 4c 4f 52 ff f0 ff fc 01 ff fd 01 70 69
72 61 74 65 0d 00 68 79 70 72 69 6f 74 0d 00 6c 73 0d 00 6d 6b 64 69 72 20
74 65 6c 6e 65 74 74 0d 00 63 64 20 74 65 6c 09 74 0d 00 6c 6c 7f 73 0d 00
76 69 20 6c 6f 67 2d 66 69 6c 65 2e 6c 6f 67 0d 00 1b 5a 5a 5b 41 ...
```

Figure 3.7: The stream of bytes with detected negotiation bytes

The rest of the data in the byte stream are byte representation of all the keys the user was typing on the keyboard. The byte stream can be split by the new line bytes which are used to send anything the user has typed to the command line to the server to process it. The split byte stream can be seen in Figure 3.8.

```
70 69 72 61 74 65
68 79 70 72 69 6f 74
6c 73
6d 6b 64 69 72 20 74 65 6c 6e 65 74 74
63 64 20 74 65 6c 09 74
6c 6c 7f 73
76 69 20 6c 6f 67 2d 66 69 6c 65 2e 6c 6f 67
1b 5a 5a 5b 41
```

Figure 3.8: The stream of bytes parsed by a new line byte

3.4 Restoring user commands

Byte representation of all pressed keys can be seen in Figure 3.8. Since there is a person sitting behind the keyboard, those bytes may be different from what was sent to the server as a command. There are two reasons for this: the users can make mistakes while typing and the users can use auto-completion to make their interaction with the server easier and faster.

3.4.1 Typographical errors

Every user typing on the keyboard makes typographical mistakes, also known as typos, from time to time. To correct a misspelt text the user has to delete the error by using the *BACKSPACE* or *DELETE* key and type the required text again.

An example of a user correcting a typo can be seen in Figure 3.9. First, the user typed `0x6c 0x6c` bytes, characters `ll` when converted from hex representation to ASCII. Then the user realized this mistake and hit the *BACKSPACE* key that can be seen in the network as `0x7f` byte. After deleting the last character, the user typed `0x73` byte that is character `s` in ASCII. The command that was sent to interact with the server is `ls`.

All four bytes which the user typed can be seen on the left part of Figure 3.9. The command that was used to interact with the server can be seen on the right side of this Figure.

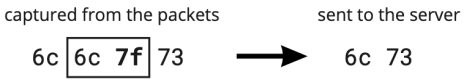


Figure 3.9: An example of a typographical error

3.4.2 Auto-completion

While interacting with the Telnet server, the user can use the auto-completion feature of the server. This feature helps users to move faster around the system without the need of remembering the complete file system architecture and all filenames.

There are two different ways of using auto-completion. The first way is using the *TAB* key while typing a command. After pressing the *TAB* key for the first time, the server completes anything that the user is typing, providing as many characters as possible. The auto-completed part of the command is then appended to anything the user has written before pressing the *TAB* key. If there are more options which can be auto-completed, the server does not complete anything. Only when the user presses the *TAB* key for the second time, the server provides all the possibilities that can be auto-completed.

The second way of using the auto-completion is by using arrows. By pressing arrow up or down the user can browse through the history of commands. When a user presses an arrow up key, the server replaces everything that was written in the command line by the previous command sent to the server.

All the bytes that were auto-completed by the server were not typed by the client. To get the auto-completed bytes, it is necessary to understand how the server-client communication works. Most of the bytes sent from the client to the server are echoed back by the server. Only after receiving those bytes on the client's side, they are displayed on the terminal screen (see Section 3.2.2). The exception when the echo bytes are different from what was sent is the auto-completion. The idea is the same for pressing the *TAB* key and arrows up or down. On the example below the auto-completion processes after pressing *TAB* key is illustrated.

After pressing the *TAB* key, the `0x09` byte is sent to the server. In most situation, the server would send back the very same byte as an echo byte. But for this byte, the server understands the command and replies to the client the completion of what the user was typing.

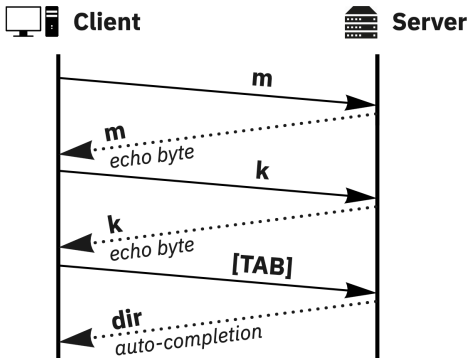


Figure 3.10: Using the *TAB* key to auto-complete the *mkdir* command

On the Figure 3.10 the client wants to create a new folder using the `mkdir` command. The user typed the `m` character, which is echoed back by the server. The same happens for the `k` character. Then the user pressed the *TAB* key. The server responds with `dir` that completes the command the user wants to use.

Both ways of auto-completion can be seen in Figure 3.11 where on the left side are all the bytes as the user typed them on the keyboard. The highlighted bytes are the *TAB* key (`0x09`) and the arrow up key (`0x5b 0x41`). On the right side of the image are the completed bytes as the user sent them to the server to interact with it.

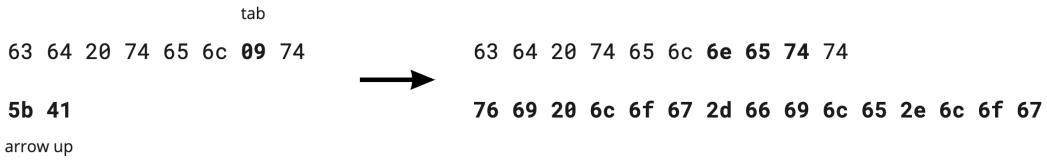


Figure 3.11: An example of auto-completion of bytes

3.5 Interpretation of user data

Obtaining all the bytes that the user typed and bytes that the user sent to the server is not providing any information about what credentials the user used. To separate credentials from the rest of the commands, it is necessary to understand when the user is typing the username and the password.

When the user connects to the server, it may ask for a username and/or password. The user provides that information only when the server asks for them, see Figure 3.12. If the provided credentials are not correct, the user is asked again to log in. If the credentials are correct, the user is successfully logged in and can interact with the server.

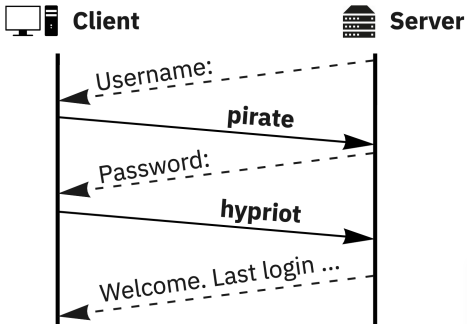


Figure 3.12: Login prompts from the Telnet server

3.5.1 Credentials

The only method of separating the user's credentials from commands is by reading and understanding what is the server asking for. Since we know the timeline of the complete connection, we can easily identify all the credentials used in the communication. The server can ask for a username using different keyword, the most used are: *Username*, *username*, *Login*, *login*, *Account*, *account*. To ask for a password, the server is usually asking for *Password* or *password*.

3.5.2 Commands

When correct credentials are provided, the user is successfully logged in and can interact with the server. Once the user is logged in, the server can send a welcome message with information about the date of the previous login. The server also sends a command-line prompt which is usually a characteristic symbol, for example `$`, `>` or `#`.

After identifying all the commands used by the user, we can process them. For the purpose of this thesis, we will use the following terminology when talking about commands used to interact with the server:

- **Command:** The complete line of text sent from the user to the server.
- **Program:** The executable part of the command, usually at the first place in the text sent to the server.
- **Options:** Additional specification how to run the program, usually starts with one dash or two dashes.
- **Parameters:** Any additional type of information given to the program, for example, a file to read, a file to write to and more.

The used terminology can be also seen in Figure 3.13.

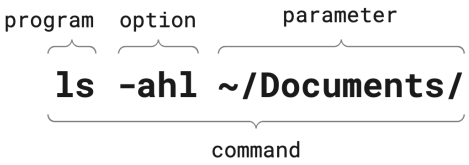


Figure 3.13: Used terminology of commands

Chapter 4

Proposed method

The method proposed in this thesis focuses on profiling Telnet users and detection of the attacks. The algorithm is divided into four consecutive steps. First, to analyse every captured Telnet packet in order to reconstruct the Telnet protocol, as mentioned in Chapter 3. Second, to create a new Telnet session object in the python code using the combination of IP addresses and ports as a key to identify each of the sessions. Third, to extract the characteristics of the typing patterns of the user, containing the used commands and the time it took the user to type them on the keyboard. Fourth, to build the user profile based on all the data extracted from the network in order to compare them.

The input data for the proposed method are files with the *pcap* format containing the captured network traffic. These captured packets have first to be processed to create Telnet sessions in order to be analysed. The method provides three different types of analysis based on how much information is extracted from the packets. The fastest analysis of captured traffic provides information about each IP address and its behaviour. The second analysis focuses on every session captured in the network and extracts all credentials used in login attempts. The most complex analysis provides a behavioural analysis of each user. The design of the method can be seen in Figure 4.1.

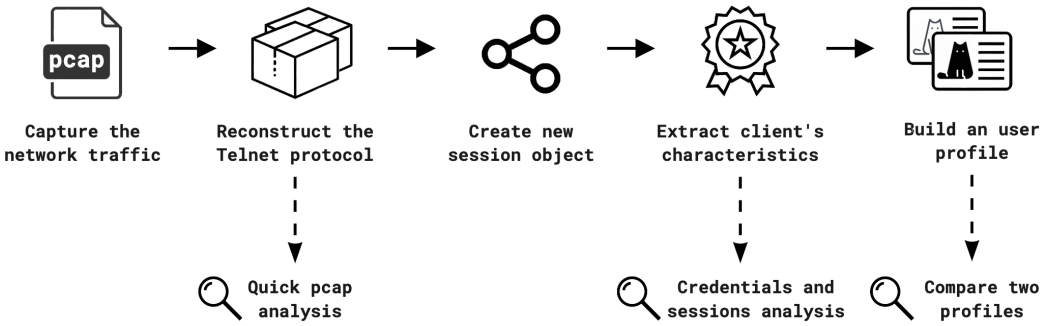


Figure 4.1: Design of the proposed method to profile and detect attacks in Telnet traffic

4.1 Create a new Telnet session

The primary data that this method works with is a network packet. Each packet is processed in order to reconstruct the TCP session and then the Telnet session. This decision was taken because there are no Telnet libraries that can provide the preprocessed data. Each TCP session is identified by the two IP addresses and the two ports using a unique key: `[client IP, client port, server IP, server port]`. Based on the source port numbers (client port) used in the communication, it is possible to identify each of the unique destination IP addresses of the Telnet server without specifying it anywhere in the program beforehand. This approach also allows the method to protect more than one Telnet device in the network. The destination port number of the server would mostly be 23/TCP or 2323/TCP (or any specified Telnet port in the program). At the same time, the client would usually use a source port defined by its operating system and that doesn't belong to the well-known or registered range of ports.

For each newly created session, the proposed method analyses all the packets that belong in it. From all the captured packets, the following information is extracted and later processed to analyse characteristics of the client's behaviour in its profile.

- **Server IP address and server port:** The identification of a Telnet server is based on its destination IP address (server IP) and destination port (server port). The proposed method can protect more than one Telnet server in the network, and based on different IP addresses of the devices, it is possible to differentiate them.
- **Client IP address and client port:** The client IP address and client port can be used to identify the client connecting to this Telnet server. If a client is connecting to a Telnet server multiple times, the source port number would, in most cases, change with every new session.
- **Start time of the session:** The date and time when the very first packet of the session was captured.
- **End time of the session:** The date and time of the last packet of the session.
- **Packet length:** The size of each packet in the session.
- **Telnet payload:** After preprocessing the Telnet protocol packets as it was explained in Section 3.3, it is possible to store details about the Telnet payload. This includes any data sent from one host to the other using the Telnet protocol. This is the most important piece of data since it gives all the details about used credentials and pressed keys.

One of the main problems of analyzing Telnet traffic in IoT data is that most malware generates thousand of Telnet connections per minute, making it very difficult to perform a quick analysis of the Telnet sessions. It can take a long time only to analyze if the Telnet connections were successful or not. Therefore different analysis schemes were proposed to speed up the results based on the number of monitored features.

The first and fastest analysis focuses on gathering statistics about all the unique client IP addresses that contacted the Telnet server. The method also provides an insight into the number of sessions per each unique IP address. The second, more in-depth analysis provides statistics about all the credentials used in the login attempts. This method creates a new user profile for each session, however, gathers only the credentials used in the login attempts. Commands and keystroke dynamics characteristics are not monitored. The last and most complex analysis focuses on behavioural characteristics of each successfully logged-in client to the server. For each session with a successful login, a new user profile is created and compared with the administrator's profile to detect any anomalous behaviour.

4.1.1 Connections statistics from all Telnet clients

The first step in the analysis of Telnet traffic is to get a high-level view of how other hosts are interacting with the Telnet server. The fastest analysis can be done by analysing all sessions coming from a single client IP address and extracting some characteristics of the payload sent in the packets. Based on the analysed data, each IP address belongs to any of these three categories:

1. **Connection to Telnet port:** There is at least one packet coming from a client to the port 23/TCP or 2323/TCP of the Telnet server in the network traffic.
2. **Sent payload:** The client has sent any payload to the Telnet server in at least one packet. The meaning of the payload in this part is not important. The payload may contain only the Telnet negotiation bytes, as well as login brute force attempts or any other user data.
3. **Interaction with the server:** The payload sent from a client to the Telnet server is analysed, and all Telnet negotiation bytes are identified and removed (see Section 3.2.1). If any bytes are not part of the negotiation bytes, the IP address of the client belongs to this category.

The results of this analysis are three lists containing IP addresses, one list for each category mentioned above. By design, any IP address that belongs to a category also belongs to all previous categories. That means that an IP address of a client that tried to log in to the Telnet server would be captured in all three lists. The comparison results of these there analysis can be seen in Figure 5.2.

Another statistics providing a better understanding of the behaviour of Telnet clients is the number of sessions. In most cases, every new session from the same client is coming from a different source port. Based on the number of different ports, it is possible to count the number of sessions coming from one IP address. From the earlier analysis of the malware network traffic, we know that some hosts are connecting to the Telnet server always from the very same port number. This port number is hard-coded in the malware source code. In this case, the only way how to capture the number of sessions is to detect when a new session starts with the TCP 3-way handshake and when it ends with the TCP 4-way handshake or by sending a packet with the *RST* bit on in the TCP header.

4.2 Create a new user profile

A new user profile is based on extracted information from a Telnet session. The stored information provides characteristics of what and how the user was typing on the keyboard. The profile stores all credentials and commands used by the user with all typographical errors, as well as the actual credentials used in the login attempt. After the client is successfully logged in to the server, they can control the device by typing commands in the command line and sending them to the server. We can create a unique profile for the each client by reading the payloads from the network, then parsing them (see Section 3.3), restoring the send payload (see Section 3.3) and understanding the meaning of the bytes sent (see Section 3.4). While processing and understanding the sent payload, the profile also extracts and stores all the special bytes used in the communication as well as typing times of all programs and commands. Moreover, the profile contains all the digraphs of typed keys on the keyboard with its time. Last but not least, the user profile contains information about when and where the session took place as well as its length and duration.

4.2.1 Payload analysis

Credentials

The user profile contains all the credentials used by the client to log in. During the login process, the user profile is filled with a list of the used usernames, another list for the used passwords, and a third list for the combination of username and password used for a single login attempt. For each of these groups, the profile contains the order in which the usernames and passwords were used and their histogram. Another captured credential feature is *how* the user was typing the credentials. The profile contains the raw bytes as they came from the network with all typographical errors as well as restored credentials that were really used for the login attempt (see Section 3.4.1).

Commands

Once the client is logged in, everything they type is considered as a command to interact with the server. Therefore the profile has the whole commands sent to the Telnet server as well as an interpretation of the executable functions, called programs (see Section 3.5.2). Similarly, as for the credentials, the order in which the commands and programs were used is stored, but not the histogram. Also, the profile contains all the raw bytes coming from the network, including every typographical error, auto-completion process and any other special byte. Finally, the profile also contains all the commands and programs interpreted as they were sent to the server.

Since our method also focuses on the characteristics of the typing patterns of users (keystroke dynamics), we tried to take advantage of this during our evaluation process. In the evaluation process with humans, we asked them to create text files in the Telnet server and to type into them. For this reason, it was necessary to implement a method to

detect when the user enters and quits text editors. All the keys typed in the text editor are considered for the keystroke dynamics. Moreover, since most text editors allow the execution of commands in the operating system, it was necessary to track these commands. The commands used in the text editor were also used as part of the commands executed by the user in the session. In the current version of the tool, the method can detect the *VI* text editor only. The detection of *nano* program or any other text editor was not implemented yet.

Typing times

For every captured byte from the network, we store the time when it was captured. Based on this data, we can extract additional characteristics of the user's typing. We implemented a keystroke dynamics method focusing on the time between typing two consecutive keys without the need of understanding the meaning of the keys. After processing all the sent payload, we store two types of times: the time of typing the whole commands, e.g. *ls -al*, and the time of typing only the program name, e.g. *ls*.

An example of all extracted values can be seen in Figure 4.2, where all the bytes were converted from the hex representation to ASCII. The non-printable `0x09` byte is displayed as *[TAB]* to show its meaning. From the raw bytes coming from the network (see Figure 4.2a), we store the time difference between capturing every two consecutive bytes. In this example, we have 10 new values. From the bytes that were sent to the server (see Figure 4.2b), we store the typing time of the command and the typing time of the program.

(a) Two consecutive bytes

(b) Command and program

Figure 4.2: Typing times captured from one command

Bytes with special meaning

One of the most essential implementations done in the proposed method is the interpretation of the special meaning of specific keys. Apart from all the standard printable keys, there are some non-printable bytes that have special meaning in the terminal. They can be divided into several groups:

- **Typographical errors:** A user can correct a typographical error by pressing the *BACKSPACE* or *DELETE* keys to remove the last typed character.
- **Auto-completion:** A user can auto-complete any text by pressing the *TAB* key. Another way of quick auto-completion is to scroll through the history of commands by using the *UP* and *DOWN* arrows keys.

- **New line bytes:** The number of times the user pressed *ENTER* in the whole session. This is a unique feature that was proposed for the first time in this thesis and that its believed to be useful in separating the sessions of users.
- **Other bytes:** The rest of the special keys, such as *ESC* used especially in the *VI* editor, $^{\wedge}C$ (*CTRL-C*) to stop commands or $^{\wedge}D$ (*CTRL-D*) to logout from sessions. The specific actions of these keys can be accomplished in a different way which creates an opportunity to differentiate users.

The number of times a user pressed each key with a special meaning provides a characteristic of how the user interacts with the server. The different captured values characterise how many typographical mistakes the user did, how much the user is using auto-completion, and also what is the user doing while thinking about the next command. Based on the user data, we know that some users are pressing *ENTER* or *ESC* key in between two commands while thinking about the next step.

4.2.2 Session information

Apart from all the bytes the client sent to the network, the user profile also contains information about the Telnet session itself. The session information stores the IP addresses and ports of both the server and the client. Based on the source IP address, the method can determine the geographical location of the client by using the *geoIP2* library that provides an offline database [10].

From the time of capturing the very first packet in the communication, the profile stores the exact time and date of when the client logged in. The profile also stores the duration of the session estimated form the time of capturing the first and the last packet. This type of information is considered metadata of the Telnet session.

Another stored characteristic is the state of the Telnet session, which characterises how the client interacted with the Telnet server. The state value can differentiate between successful and unsuccessful logins by interpreting and understanding the prompts coming from the server. The state value also stores information about how the session was ended - either by following the TCP 4-way handshake or by sending a TCP packet with the *RST* bit on.

The last stored characteristic of the session is its total size in bytes. This value is obtained by the sum of the length of every packet captured in the network. Every captured packet is counted in this characteristic regardless if it's carrying any payload or not, or if it was re-transmitted or not.

4.2.3 Bot detection

The proposed method can also differentiate if the client of a Telnet session was an automated tool or a human user. The algorithm to find this difference is based on the analysis of times while typing two consecutive bytes. This analysis is also done for programs and commands sent by the client.

Every key that the client types on the keyboard in the Telnet communication is immediately sent to the server. Therefore, if the client is a person typing on the keyboard, the typing times for two consecutive bytes are bigger than 0. The typing times of the commands and programs are at least in some cases bigger than 0 as well. There is one particular case of times between bytes being zero, and that is when the user is pasting the text into the command line or if they use the auto-completion features, such as scrolling through the history of commands.

On the other hand, a bot executes predefined commands and all the bytes are sent together in one packet. Therefore, all the typing times are equal to 0. Moreover, since all the commands were predefined, they don't contain any typographical errors or corrections.

4.3 Compare two profiles

After the method creates a new profile for each new session, its possible to compare all the profiles between each other. Each profile contains several data structures which need to be processed in a different way depending on the meaning of the stored information. Each type of structure needs a different kind of comparison. In user profiles, there are different types of structure:

- **Integer, float number:** The size of each session and the day when it started are stored as *integers*. The duration of a session is stored as a *float*, as well as the time of when the session started.
- **String:** The IP addresses of both hosts in the session are extracted from the packets and stored as *string* data structure.
- **List of strings:** All usernames, password and commands are extracted as *string* structures and stored in a list following the order in which they were sent to the network.
- **Python counter:** After processing all data sent to the network, the profile stores a histogram of all usernames and passwords used in login attempts, as well as the combination of username and password. The *counter* data structure is also used to store characteristics about the usage of special bytes.
- **Python dictionary:** Each command and program used in the communication is added to a *dictionary* data structure as a key with a list of typing times as a value. The keystroke dynamics statistics are also stored as *dictionary* with two consecutive bytes as the key and list of typing times as the value.

4.3.1 Distance between two profiles

Once that the different type of structures inside the profile were explained it is possible to show how each of these structures is compared. The general algorithm to compare two profiles is the following. For each feature in the profiles:

1. Get the value of that features for both profiles
2. Compute a distance between the features, according to the type of the feature (integer, string, list of strings, etc.)
3. Normalize the distance between features using the Min-Max scaling method.
4. Apply weight to each feature comparison according to a heuristic training.
5. Sum all the weighted distances into a final distance between the profiles.

The Min-Max scaling method is defined as:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.1}$$

where x_{min} is the minimum value a feature can take and x_{max} is the maximum value a feature can take. The x_{max} and x_{min} values used for each individual feature are described below and in Table 4.1.

Instead of computing the similarity between profiles, the proposed method computes the distance between each feature between two profiles. If the normalized value for any distance of any feature equals 0, it means that this feature is the most similar between these two profiles. If the normalized value equals 1, the feature is the most distant. The same concept applies to the final distance between profiles. A distance of 0 means that both profiles are the most similar, and a distance of 1 means that both profiles are the most different.

Feature	x_{min}	x_{max}	**Description**
Session size	0	5 000 000	= 4.7 MB
Session length	0	9 000	= 2.5 hours
Login day of year	0	365	= 1 year
Login day in month	0	30	≈ 1 month
Time in a day	0	86 400	= 24 hours
IP address	0	4 294 967 295	$= 2^{32} - 1$

Table 4.1: The minimum and maximum values of each normalized feature in a profile

4.3.2 Distance between individual profile feature

This section describes how each feature was compared in order to obtain the final distance between user profiles. Each part includes the algorithm of how the features were compared along with the details of the comparison algorithm.

Session size and length

The size of the session is defined as the sum of sizes of each packet in the session. This value is computed and stored in bytes and considers all packets sent in the communication regardless of their meaning. The length of the session is calculated as the time difference between the last and first packet captured in the session.

Both of these values are stored in the profiles as numbers. The difference between them is calculated as the absolute of subtracting the value from each profile:

$$x_{dist} = |x_1 - x_2|,$$

where x_1 is the stored value in one profile, x_2 is the stored value in the other profile and x_{dist} is the final distance.

The minimum value for computing the normalized distance of the session size is 0, meaning that the size of the two sessions is the same. The maximum value is set up to 5 000 000 bytes (= 4,77 MB). For calculating the normalized value of session length, the minimum value is also equal to 0, meaning that the length of the two sessions is the same. The maximum value is 9 000 seconds (= 2,5 hours). The values for the maximum of these features was heuristically selected based on expert information on common Telnet usage.

Day and time of login

Each profile stores the information about the day and time of a successful login as two separate values. Profiles stores the day in a year of a successful login as a separate value from the time of the login on that day. The distance of this feature between two profiles can be computed by using the following equation:

$$x_{dist} = |x_1 - x_2|,$$

where x_1 is the stored value in one profile, x_2 is the stored value in the other profile and x_{dist} is the final distance.

To calculate the normalized value for the difference in which day of the year the user logged in, the minimum value is set to 0, meaning the two sessions took place on the very same day. The maximum value is set up to 365 days, that is equal to 1 year. The method uses the following formula to obtain an approximate day in a month when the user logged in:

$$x_{month} \equiv x_{year} \pmod{30},$$

where x_{year} is the absolute difference between the login day and x_{month} is the difference between the login day in a month.

The difference in the time of login in a day is calculated in seconds. The minimum value is 0, meaning the two sessions started at the very same time in a day. The maximum value is set up to 86 400 seconds (= 24 hours).

Client IP address

Since the proposed method can analyse all packets in the network, including the ones coming from the Internet, it is necessary to understand if the client is connecting from a local network or not. For this purpose, the method can analyse each IP address and determine if it belongs to the internal private range of IP addresses (10.0.0.0/8, 172.16.0.0./12, 192.168.0.0/16) or not.

While comparing the IP addresses of two Telnet sessions, if only one IP address is a private IP and the second one is a public IP address, then the comparing method returns the maximum distance between these two IP addresses. But if both IP addresses are public, or both are private, the method returns as a distance the number of IP addresses in between these two.

To calculate the normalised distance between two IP addresses, the method uses 0 as the minimum value, meaning the client is connecting from the very same IP address. The maximum value is set up to 4 294 967 295, which is the total number of IPv4 addresses.

Used commands

All the commands that the client used in the communication are stored in a python list. The profile stores all the commands as they came directly from the network as well as the processed and restored commands after interpretation. To compare such lists, the proposed method is using two different comparing algorithms.

The first comparing method is the sequence matcher [11] algorithm used for comparing pairs of sequences of any type. The algorithm is based on the Ratcliff/Obershelp pattern recognition [6] which searches for the longest contiguous common sub-sequence present in both lists.

The second method used to compare two lists of strings is the algorithm of computing the Levenshtein similarity ratio based on the Levenshtein distance. The Levenshtein distance computes the number of single-character edits required to change one string to the other. These changes are insertion, substitutions and deletions. [22] To compute the similarity ration, our proposed method uses the *python-Levenshtein* library [5] providing the similarity between two lists.

Both methods compute the similarity between two lists in the range of $\langle 0, 1 \rangle$. To get the distance between two lists, we compute the complement of the similarity value

$$x_{dist} = 1 - x_{sim},$$

where x_{sim} is the similarity ratio between two lists of strings and the x_{dist} is the final distance value.

The usage of special bytes

Each profile contains a counter to keep the histogram of special characters used in the communication. The key in this counter is any unprintable character used in the communication or any recognised byte combination with any special meaning, for example, the arrow *UP* key.

To compare counters from two profiles, we compute the distance based on the cosine similarity [28]. This method is based on measuring the cosine of the angle between two vectors and returns the similarity in the range $\langle 0, 1 \rangle$. The cosine similarity of two counters A and B can be computed as

$$cos(A, B) = \frac{\sum_{i \in A \cap B} a_i b_i}{\sqrt{\sum_{i \in A} a_i^2} \sqrt{\sum_{i \in B} b_i^2}},$$

where $a_i \in A$, $b_i \in B$, and A, B are dictionaries to be compared. To obtain the distance between two counters, we compute the complement to the cosine similarity

$$x_{diff} = 1 - cos(A, B).$$

Typing times

To capture the typing times, the profile contains three separate dictionaries, one for the typing times of commands, the second one for typing times of programs and the last one for typing times between each pair of pressed keys. Each dictionary contains the sent command (or program or two consecutive bytes) as a key and a list of typing times for each time that the key was used. For example, if the key pair *AD* was pressed three times, the values for the key *'AD'* may be *[0.34, 0.11, 0.56]*.

The comparison algorithm is based on the Student's t-test statistical test. For each list of typing times from the dictionaries stored in the profile, the method uses the Student's t-test to estimate the probability that the two lists of values came from the same probability distribution. [15][40]

The null hypothesis that is being tested is: *Both lists of typing times were created by the same user.* After running the test, the calculated p-value is considered as a distance between the two lists and used for computing the final distance between two profiles.

If one profile contains only one typing time for a key while the other profiles contain more than one time value, the method will perform the one-sample t-test. This method compares the mean of a group of data to a single known value. [14]

In case that there is only one captured time value for the same key in each profile, the Student's t-test cannot be used. The difference between them can be computed as the complement of their ratio.

$$p_i = 1 - \frac{min(a_0, b_0)}{max(a_0, b_0)},$$

where a_0 and b_0 are the time values from each profile for the same key i and p_i is the distance between them. The final distance between two dictionaries is then computed as a product of all partial distances.

$$x_{dist} = \prod_i p_i$$

4.3.3 Final distance between two profiles

The total distance between two profiles can be computed as a weighted linear combination of distances between each feature. The total distance between the two profiles can be computed as follows:

$$d = \sum_i w_i x_i, \tag{4.2}$$

where w_i is the weight for feature i and x_i is the distance between feature i. The weights used for this algorithm can be seen in Table 4.2. All the weights were set up heuristically by experts (see Chapter 5).

Feature i	**Algorithm**	**Weight** w_i
Duration	difference	0.02702702703
Length	difference	0.05405405405
IP address	difference	0.02702702703
Start time	difference	0.09459459459
Day in a year	difference	0.1351351351
Day in a month	difference	0.1351351351
Raw commands	Sequence matcher	0.06756756757
Raw commands	Levenshtein distance	0.05405405405
Processed commands	Sequence matcher	0.06756756757
Processed commands	Levenshtein distance	0.05405405405
Special bytes	Cosine distance	0.08108108108
Typing times of commands	t-test	0.06756756757
Typing times of programs	t-test	0.06756756757
Typing times of two consecutive bytes	t-test	0.06756756757
	$\sum_i w_i$	1.0

Table 4.2: The weights used for each feature to compute the final distance

Chapter 5

Results

This chapter provides the results of the proposed method on the captured Telnet traffic going to a monitored Telnet server. The server was running for a total of 70 days, from 17 October 2019 to 25 December 2019 and the traffic was monitored and stored all the time. In the first section, we describe how we created a monitored infrastructure with the Telnet server. The following section provides results from the quick analysis of the Telnet traffic, focusing on the behaviour of unique IP addresses and all the sessions in the captured traffic. The next section analyses all the credentials used by clients to tried to log in into the monitored Telnet server. The following section describes the creation of all the user profiles created for the purposes of this thesis. The last section provides the results of comparing the user profiles and the distances computed between them.

5.1 Experiments setup

To create accurate datasets to test the proposed method, we set up a Telnet server in a monitored infrastructure. The Telnet server was running on a Raspberry Pi 3B+ computer with installed HypriotOS [12]. There were two opened ports in the device, the Telnet port 23/TCP and the SSH port 22/TCP. The SSH port was used to set up a Telnet server and to prepare user account for testing the proposed method. The default credentials of the HypriotOS were changed in order to prevent any infection getting into this device and possibly compromise the results.

The Raspberry Pi was part of the infrastructure created and maintained by the Aposemat research group [7], which focuses on the analysis of IoT malware. As part of their research, they infect devices with different IoT malware families to observe their behaviour. For this reason, the Internet speed in the monitored network is limited to a maximum of 500 kbps per each device. The maximum speed for all the connected devices together is limited to 10 Mbps. The number of connected and active devices in the monitored network varies according to the ongoing research of the Aposemat research group.

The infrastructure of the Telnet server connection can be seen in Figure 5.1. The Telnet server is connected to a switch with a port mirroring option turned on. Every packet going

through this switch is mirrored to a different port where it is captured into a **.pcap* file. This switch is connected to a router with a static public IP address providing the Internet connection. All the traffic going to this public IP address on port 23/TCP is redirected to the Telnet server. This router is located at the edge of the monitored network, and the bandwidth limitation is set up in it.

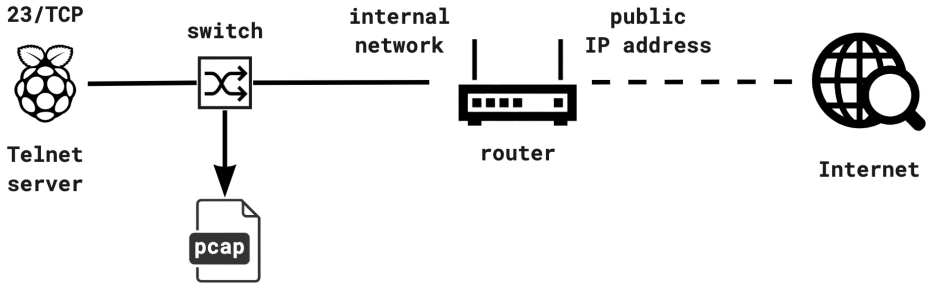


Figure 5.1: The infrastructure to capture data

The traffic going to and from the Telnet server is continuously monitored and stored into **.pcap* file. A new **.pcap* file is created and saved to the storage every 24 hours. In order to organize all the **.pcap* files, the name of each of them consists of the date and time of starting the 24-hour monitoring and the internal IP address of the monitored Telnet server.

5.2 Analysis of IP addresses from every client

A high-level understanding of the network traffic can be done by analysing the behaviour of every unique IP address connecting to the Telnet server. The proposed method keeps statistics of the individual IP addresses in each provided **.pcap* file. All the results described below comes from the analysis of the network traffic captured in 24-hour time windows. The change in the number of unique IP addresses can be seen in Figure 5.2.

The total number of unique IP addresses that connected to the Telnet server in 70 days is 10 384. In every 24 hours, the Telnet server was on average contacted by a new unique IP address 178 times. In other words, the Telnet server was approached by a new unique IP address every 8 minutes. In this group of IP addresses are all the hosts that sent at least one packet to the Telnet server, including the Internet scanners.

Out of all the IP addresses contacting the Telnet server, there is on average 54% of them reaching the server only on the Transport layer. All those IP addresses never sent any payload and therefore never interacted with the server on the Application layer. The remaining 46% of all IP addresses sent some payload to the Telnet server in order to interact with it.

Further analysis of the payload coming from each IP address provides an insight into how each IP address interacted with the Telnet server. In the group of IP addresses sending some payload, there is 49% of them sending only the Telnet negotiation bytes. All these hosts were real Telnet clients that followed the Telnet protocol, but they never send any

other user data. The additional 51% of IP addresses sent bytes that were not part of the Telnet negotiation bytes. This payload interacted with the server by entering the credentials or sending commands.

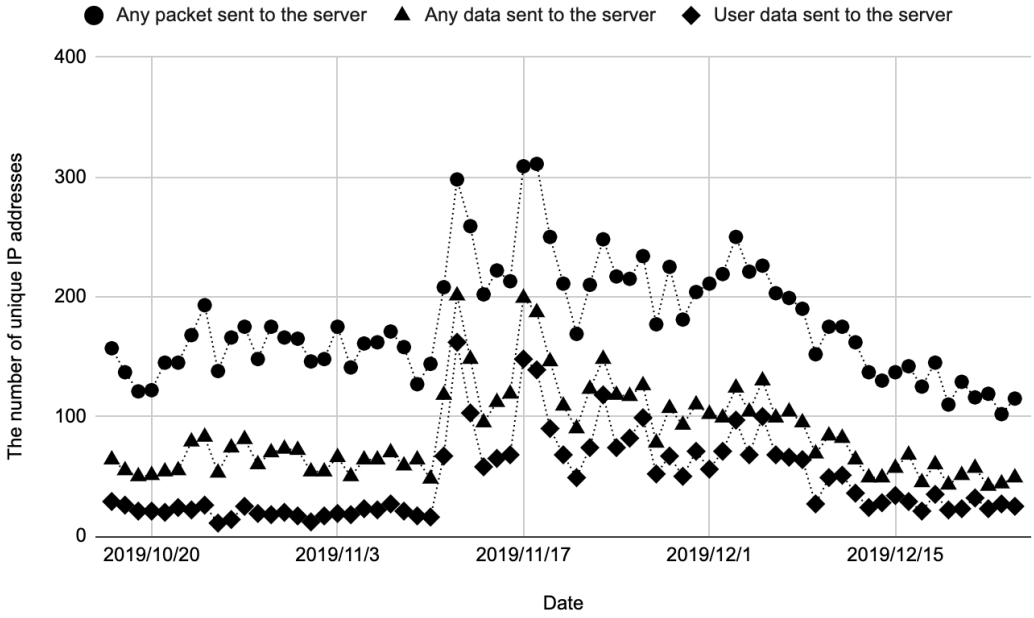


Figure 5.2: The number of unique IP addresses interacting with the Telnet port

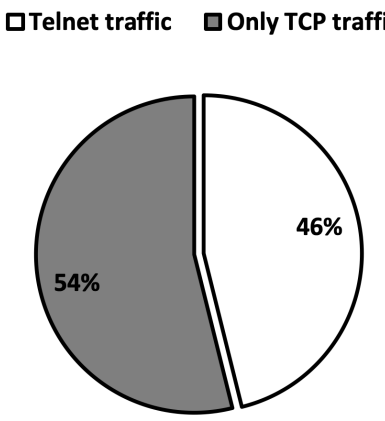


Figure 5.3: The number of unique IP addresses sending only TCP traffic and sending also Telnet traffic

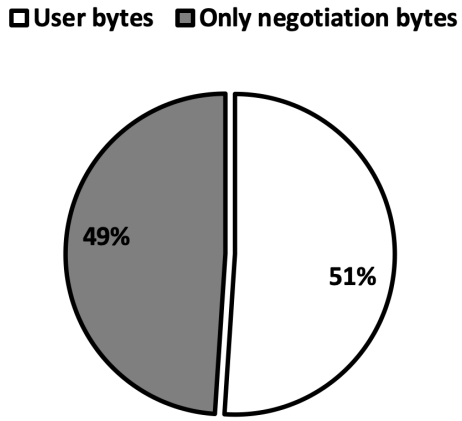


Figure 5.4: The number of IP addresses sending only the Telnet negotiation bytes and sending the user data bytes in 24 hours

5.2.1 Analysis of sessions

A better view on how many times the Telnet server was contacted provides the number of sessions started with the server. The average number of sessions in a 24-hour time window was 1400. The number of sessions reached a minimum of 698 on 17 December 2019 and the maximum of 2868 on 5 November 2019. The median of all measured values was 1292. The results of the analysis can be seen in Table 5.1.

Min	698
Max	2 868
Mean	1 400
Median	1 292

Table 5.1: The number of sessions per 24 hours

More interesting statistics provide the number of sessions per unique IP address captured from the network. On 5 November 2019, we captured the maximum number of sessions originating from a single IP address. There were 1 237 sessions coming from the *104.248.49.211* IP address registered to DigitalOcean, LLC hosting cloud services. This outlier caused the average number of sessions per IP address to be 9. Although the median is 2, which means that at least half of all the IP addresses made at most 2 individual sessions to the Telnet server during the monitored time. The results are shown in Table 5.2

Min	1
Max	1 237
Mean	9
Median	2

Table 5.2: The number of sessions per IP address

5.3 Analysis of credentials

From all the captured traffic, the method can extract credentials used for login attempts. During the whole capturing time, we recognised 294 unique combinations of user and passwords sent to the Telnet server during the login process. On average there were 2 260 login attempts in 24 hours.

The method can process all the username and password combinations and provide their histogram. The list of most used credential combinations can be seen in Table 5.3, more detailed list can be seen in Appendix A. Out of all the used username and password combination, in 44.65% of all cases, the client used a username `root` with no password. After

Username	Password	Usage	Usage %
root		70 853	44.76%
admin	1234	1 941	1.23%
root	aquario	1 940	1.23%
ubnt	ubnt	1 146	0.72%
root	root	1 052	0.66%
admin	admin	1 017	0.64%

Table 5.3: The most used credential combinations to login to the Telnet server in 70 days

closer investigation of the Telnet traffic, we discovered a strange behaviour of the Telnet server.

For most of the session, the Telnet server is first asking for the username of the client. All the bytes sent by a client are sent back to the client as echo bytes. Then, the Telnet server asks for a password. Any bytes the client sends to the server are not echoed back. When the provided credentials are incorrect, the server informs the client, and the login process starts again until the maximum number of attempts is reached.

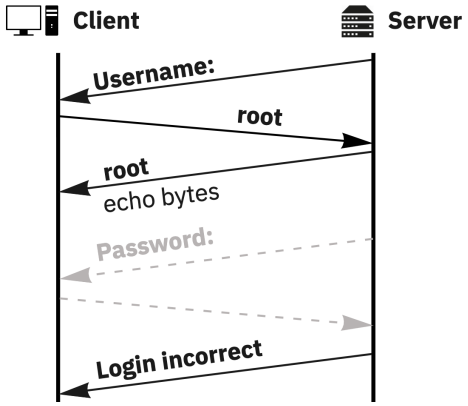


Figure 5.5: A strange behaviour of the Telnet server

In the 44.76% of all login attempts, the Telnet server first asks for a username. The client sends the username `root` and the server echoes the username back. But in the next step, the Telnet server didn't ask for a password. Instead, it sent the *Login incorrect* information and restarted the login process. The behaviour is shown on the Figure 5.5, where the missing password prompt is displayed in dashed line in grey colour.

We manually researched sessions where the password prompt is missing to understand what happened. Based on the sequence numbers in the TCP header of each packet, we know that we captured all the traffic between the client and the server. We also know that the root user in Telnet server is not blocked in any way, since we can see other credential combinations with root as a username and some password, for example `root\aquario` or

`root\root`. Based on our investigation we couldn't understand why we see this strange behaviour of the Telnet server, and we couldn't replicate this behaviour. Therefore the reason why we see this anomalous behaviour is unknown.

The histogram of all used credentials can also provide an insight into other attacks against the Telnet server that are not based on brute-forcing the username and the password. In this category of send payload are commands which are relevant to be sent to the Telnet server, for example `shell`, `sh` or `enable`. Those commands try to interact with the Telnet server and they would be successful if the Telnet server was accessible without any credentials. The fact that we captured those attempts indicates that there are Telnet devices connected to the Internet without any username and password protection.

In the group of the least used credentials, we can also find attacks against other services. An example of such an attack can be seen in Figure 5.6, where a client is sending an HTTP GET request to port 23/TCP. This kind of payload is not really unexpected since the administrator of the server can run any service on any port. Therefore it is normal to see for example HTTP request on Telnet port.

```
GET / HTTP/1.1
Host: 147.32.xx.xxx:23
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/66.0.3359.117 Safari/537.36
Accept: */*
Accept-Encoding: gzip
```

Figure 5.6: A HTTP request received on Telnet port

The last category of the least used credentials are strings with no apparent meaning. An example of those strings converted from hex representation to ASCII can be seen in Figure 5.7.

```
0e09f664c9cf8229aade9c16d6d78560a71ac02fc02bc011c007c013c009c014c0

03134e642b1803b5f180e3d830de1107c3e0901ac02fc02bc011c007c013c009c014c0

1603019a019603033a29aedbc6cc348b45bebcf1f190

c02fc02bc027c023c013c009a29e674033329a994544c031c02dc029c025c00ec0049c3c2f9
641c011c007c00cc0020504151209016d0b04030102
```

Figure 5.7: Strings captured as credentials with unknown meaning

5.4 Creating new profiles

To create datasets for testing the proposed method, we asked several users to participate in user testing. Each user was given the same credentials for the Telnet server and some tasks to do in the server. The tasks and guidelines are shown in Appendix B. To make sure

that we capture enough data for user profiling, we asked users to log everything they find while working on the tasks into a text file.

The tasks that were given to the users were meant as guidelines to make sure that users create a long enough profile. The user could work on the given tasks in any order using any commands. This testing was not meant to be an examination of the user's skills. Therefore users were allowed to use any source to find out a correct answer, including asking a colleague or searching on-line.

In each task, the user was asked to create a new folder with a given name. In case that this folder already existed in the server, the users were encouraged to improvise. Some users created a new file or folder with a similar name, some users created a new file or folder with a completely new name, and some users didn't create anything and worked with what was already in the server.

Apart from following the given guidelines, the users were allowed to do arbitrary actions in the server. To prevent any malicious activity, we asked users not to infect or attack any device in the network, internal or external. The full guidelines that were given to each user can be found in Appendix B.

Since the decision making of this method is not based on the used credentials, we decided to use all the captured sessions in the datasets. In the Telnet server were two user accounts, one used for the users to work on the given tasks and the other account for administrating the device.

After running the proposed method on all the captured traffic, the tool recognised and stored 32 different user profiles. The dataset contained 27 profiles created by 4 different users. The remaining 5 user profiles that were generated were not created from a successful login session. The distribution of datasets can be seen in Table 5.4.

Number of profiles	**Description**
14	Profiles of user 1
5	Profiles of user 2
7	Profiles of user 3
1	Profile of user 4
4	Profile containing data with unknown meaning
1	An empty profile

Table 5.4: The user profiles generated by the proposed Telnet analyser

We asked computer science students and professionals to participate in the experiment. The goal was to find users who are skilled in typing on the computer keyboard and fluent in English. However, the users had a different experience in working with the terminal and connecting to the Telnet server. All users used their own computers to work on the given assignments, and therefore we expect them to be used to typing on their keyboard.

In 4 cases the method falsely generated a user profile for a bot sending some data to the server. The last stored user profile did not contain any data. All these profiles were created because of a bug in the source code, and therefore they can be dismissed.

5.4.1 Differentiate bots from humans

The correct username and password for logging in to the Telnet server were on purpose set up to non-trivial ones. Our goal was to prevent anyone from infecting the device and to capture only the relevant user profiles. However, this prevention doesn't necessarily mean that an unauthorised user didn't try to log in to the Telnet server by manually brute-forcing the credentials.

A simple algorithm based on capturing the typing times can differentiate between a person typing on the keyboard and an automated bot. In all the network traffic we captured in 70 days, there were only log in attempts made by users we asked to participate in our research and the automated bots that are active on the Internet. No other person tried to log in to our Telnet server.

The simple algorithm of separating bots from humans has its limitations. The humans can be detected only if they type on the keyboard. If the human would prepare all the commands before connecting to the server and then copying them and pasting to the terminal, the method would not detect them. An automated tool may pretend to be a human being if it would imitate the typing on the keyboard. The analysis of known malware proves that the design of the malicious code is very simple. Therefore nowadays we don't expect such advanced malware to be spreading over the Internet.

5.4.2 Comparison between profiles

A new user profile is created for each session that contains a successful login; however, not all the profiles can be used in our method. From all the profiles created from the method, two user profiles had to be discarded. One profile of the user 1 contains the credentials and only one command, and one profile of the user 2 contains the credentials and only two commands. Both those profiles do not contain enough information to make a decision about the user. For that reason, the proposed method would only inform the administrator of the network, and discard the profiles.

After discarding all the unimportant profiles from our dataset, there were 25 profiles of 4 different users left. We compare each of them with all the other user profiles. The total number of comparisons was 310. Every user profile used in the comparison was labelled with the identification number of the user.

Since there are several users, the comparisons between profiles can be of two types: the profiles came from the same user, or the profiles came from different users. When the profiles came from the same user, the proposed method should give a small distance. When the profiles came from different users, the proposed method should give a large distance.

To calculate the final distance between every pair of user profiles, we used the method described in Section 4.3.3. We sorted all user profiles pairs in descending order by their distances. The final distances for all the comparisons are in the range from 0.1340104555 to 0.6276173244. In the sorted list of all compared pairs of user profiles, we could see that the distances between two user profiles of the very same user are lower than the distances between two user profiles of two different users. The average distances between users can be seen in Table 5.5. Since the user 4 provided only one Telnet session containing all the testing scenarios, all the distances between this user and all the other users are based only on one comparison.

	User 1	**User 2**	**User 3**	**User 4**
User 1	0.348492	0.413170	0.413376	0.389913
User 2	0.413170	0.291622	0.436204	0.347900
User 3	0.413376	0.436204	0.279553	0.356700
User 4	0.389913	0.347900	0.356700	0

Table 5.5: The average distance between profiles of each user

Based on all the computed distances, we could set up a distance that would separate the user profiles that were created by the very same users from the user profiles created by two different users. We set up this boundary distance to 0.3060, where any final distance lower than this value means that the two profiles were created by the very same user. If the final distance between the two profiles is larger than this distance, we can conclude that the two profiles were created by two different users.

The number of user profile pair that were created by the very same user and they were correctly classified was 49, the number of user profiles pairs classified incorrectly was 13. On the other hand, the number of correctly classified user profiles created by two different users was 186, and the number of incorrectly classified pairs was 62. The confusion matrix can be seen in Table 5.6.

	True Positive	**True Negative**
Predicted Positive	49	13
Predicted Negative	62	186

Table 5.6: The confusion matrix

Data from the confusion matrix can be used to compute additional measures of the proposed method. The computed values can be seen in Table 5.7.

The measured data proves that the authentication method based on the user's behaviour is possible under certain conditions. There are plenty of circumstances to take into account, but the differences can be large enough to be useful. The proposed method in this thesis can be improved by using a better approach and better algorithms to set up weights of features

used for computing the final distance.

Measure	**Value**
True positive rate	0.441441
False positive rate	0.065326
Precision	0.790323
Accuracy	0.758065
F1 score	0.566474

Table 5.7: The final measures computed from the confusion matrix

Chapter 6

Conclusion

The problem of the amount of devices with weak or default credentials connected to the Internet is getting bigger. All those devices are vulnerable to any attack, and usually, it doesn't take long until they get infected. Many of these devices are the IoT devices that are getting more popular in everyone's home. Many of the cheap devices that users can buy nowadays are using the Telnet protocol. Even when the Telnet protocol is still being used, there was no automatic tool that would analyse the Telnet traffic captured from the network.

The first part of this thesis focused on understanding the Telnet protocol and finding a way how to extract the communication automatically. This part provided all the complications that the method has to deal with and the solutions to those problems. The second part focused on creating a tool that can extract and analyse all the payload send between devices.

The proposed method was introduced with different levels of analysis. The tool can provide statistics about the number of IP addresses and the number of sessions connecting in a **.pcap* file. From the Telnet payload, the tool can extract all the credentials and commands used by the client. Based on the behaviour of the client, the method introduced a way of how to create a user's profile, compare the behaviour of two clients and decide if the user is the same or not.

In the future, the method can be extended to read the network traffic in real-time directly from an interface instead of processing only the **.pcap* files. The technique would then be able to notify the administrator of the network about any anomalies in the network as soon as they get detected. The method may be later further extended to create a continuous authentication of a user and provide better protection for every user.

The results of the provided method proved that it is possible to create user profiles and compare them in order to authenticate the client. We created user profiles of 4 different users connecting to our Telnet server 27 times in total. The proposed method works with a precision of 0.79, accuracy 0.76, FPR 0.07, TPR 0.44 and F1 score 0.57. We believe that the decision algorithm can be improved by further analysis of the clients' behaviour.

In the next version of the tool, we would like to achieve better time and space complexity

of the implementation. This step would require reimplementation of the important parts of the systems and changing most of the architecture of the system. Therefore this action is left for future work.

In the proposed method, we have implemented many functions for reading and analysing the Telnet traffic directly from the network. We would like to share this part of the tool to the community as a Python library for Telnet analysis.

Based on our research, in most of all cases, the devices connected to the Internet are being attacked by guessing the correct credentials. After all, we can still see the Mirai botnet spreading over the Internet. The brute-forcing method will be effective until all users become aware of the security of their network devices. Only after people stop using default or easy to guess credentials, the security of every device in the network will increase.

Bibliography

[1] 'Creating Traffic Profiles'. https://www.cisco.com/c/en/us/td/docs/security/firesight/541/user-guide/FireSIGHT-System-UserGuide-v5401/Traffic-Profiles.pdf. Accessed: 2019-11-17.

[2] KrebsOnSecurity Hit With Record DDoS. https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/, 2016. Accessed: 2019-10-22.

[3] Who Makes the IoT Things Under Attack? https://krebsonsecurity.com/2016/10/who-makes-the-iot-things-under-attack/, 2016. Accessed: 2019-10-22.

[4] MMD-0056-2016 - Linux/Mirai, how an old ELF malcode is recycled.. https://blog.malwaremustdie.org/2016/08/mmd-0056-2016-linuxmirai-just.html, 2016. Accessed: 2019-10-22.

[5] Python-Levenshtein library. https://pypi.org/project/python-Levenshtein/#documentation, . Accessed: 2019-12-12.

[6] Ratcliff/Obershelp Pattern Recognition. https://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html, . Accessed: 2019-12-12.

[7] Aposemat Project. https://www.stratosphereips.org/aposemat. Accessed: 2019-11-13.

[8] DevFest.cz 2019. https://2019.devfest.cz/home. Accessed: 2019-11-17.

[9] Internet of Things Forecast. https://www.ericsson.com/en/mobility-report/internet-of-things-forecast. Accessed: 2019-11-17.

[10] GeoIP2. https://dev.maxmind.com/geoip/geoip2/, . Accessed: 2019-11-26.

[11] difflib. https://docs.python.org/3/library/difflib.html, . Accessed: 2019-12-12.

[12] Hypriot. https://blog.hypriot.com/. Accessed: 2019-11-30.

[13] OWASP Czech Republic. https://www.owasp.org/index.php/Czech_Republic. Accessed: 2019-11-17.

[14] One Sample T Test: How to Run It, Step by Step. https://www.statisticshowto.datasciencecentral.com/one-sample-t-test/, . Accessed: 2019-12-14.

[15] T-test using Python and Numpy. `https://towardsdatascience.com/inferential-statistics-series-t-test-using-numpy-2718f8f9bf2f`, . Accessed: 2019-12-14.

[16] Argus. `https://openargus.org/`, . Accessed: 2019-11-19.

[17] Shodan. `https://www.shodan.io/`, . Accessed: 2019-11-19.

[18] Snort - Network Intrusion Detection & Prevention System. `https://www.snort.org/`, . Accessed: 2019-11-19.

[19] Suricata - Open Source IDS/IPS/NMS engine. `https://suricata-ids.org/`, . Accessed: 2019-11-19.

[20] The Zeek Network Security Monitor. `https://www.zeek.org/`, . Accessed: 2019-11-19.

[21] BERA, A. 80 IoT statistics. `https://safeatlast.co/blog/iot-statistics/`, 2019. Accessed: 2019-11-13.

[22] BEREZOVSKY, M. – MARIKK, R. Pokrocila algoritmizace, Text Search, 2012.

[23] BORMAN, D. Telnet Authentication Option. RFC 1409, RFC Editor, January 1993. Dostupné z: `https://www.rfc-editor.org/rfc/rfc1409.txt`.

[24] BORMAN, D. Telnet Authentication Option. RFC 1416, RFC Editor, February 1993. Dostupné z: `https://www.rfc-editor.org/rfc/rfc1416.txt`.

[25] CARR, C. S. Network Subsystem for Time Sharing Hosts. RFC 15, RFC Editor, September 1969. Dostupné z: `https://www.rfc-editor.org/rfc/rfc15.txt`.

[26] DEMETER, D. – PREUSS, M. – SHMELEV, Y. IoT: a malware story. `https://securelist.com/iot-a-malware-story/94451/`, 2019. Accessed: 2019-11-19.

[27] DOUHOU, S. – MAGNUS, J. R. The reliability of user authentication through keystroke dynamics. *Statistica Neerlandica*. 2009, 63, 4, s. 432–449.

[28] HAN, J. – PEI, J. Cosine Similarity. *Data Mining: Third Edition*. 2012.

[29] JADHAV, C. et al. Biometric authentication using keystroke dynamics. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, s. 870–875. IEEE, 2017.

[30] KASIMOV, Y. Anomaly detection of host roles in computer networks. Master's thesis, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Czech Republic, 2018.

[31] KHARE, R. Telnet: the mother of all (application) protocols. *IEEE Internet Computing*. 1998, 2, 3, s. 88–91.

[32] KUBESA, D. Identification of network users by profiling their behavior. Master's thesis, Czech Technical University in Prague, Jugoslávských partyzánů 1580/3, 160 00 Prague, Czech Republic, 2018.

[33] KUMAR, A. – LIM, T. J. EDIMA: Early Detection of IoT Malware Network Activity Using Machine Learning Techniques. *arXiv preprint arXiv:1906.09715*. 2019.

[34] KUMAR, D. et al. All things considered: an analysis of IoT devices on home networks. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, s. 1169–1185, 2019.

[35] KUZIN, M. – SHMELEV, Y. – KUSKOV, V. New trends in the world of IoT threats. `https://securelist.com/new-trends-in-the-world-of-iot-threats/87991/`, 2018. Accessed: 2019-10-22.

[36] LEHTINEN, S. – C. LONVICK, E. The Secure Shell (SSH) Protocol Assigned Numbers. RFC 4250, RFC Editor, January 2006. Dostupné z: `https://www.rfc-editor.org/rfc/rfc4250.txt`.

[37] LIBIGER, M. Zlín pořídil chytré odpadkové koše. Mají lis a samy ohlásí, že jsou plné. `https://www.idnes.cz/zlin/zpravy/odpadkove-kose-lis-komunikace-naplnenost-zlin.A191107_512684_zlin-zpravy_ras`, 2019. Accessed: 2019-11-13.

[38] MAHMOOD, H. B. Transport layer security protocol in Telnet. In *9th Asia-Pacific Conference on Communications (IEEE Cat. No. 03EX732)*, 3, s. 1033–1037. IEEE, 2003.

[39] MONROSE, F. – RUBIN, A. Authentication via keystroke dynamics. In *Proceedings of the 4th ACM conference on Computer and communications security*, s. 48–56. Citeseer, 1997.

[40] NAVARA, M. *Pravděpodobnost a matematická statistika*. České vysoké učení v Praze, 2007.

[41] PATEL, M. – SHANGKUAN, J. – THOMAS, C. What's new with the internet of things? `https://www.mckinsey.com/industries/semiconductors/our-insights/whats-new-with-the-internet-of-things`, 2017. Accessed: 2019-10-22.

[42] POSTEL, J. – REYNOLDS, J. TELNET PROTOCOL SPECIFICATION. RFC 854, RFC Editor, May 1983. Dostupné z: `https://www.rfc-editor.org/rfc/rfc854.txt`.

[43] SAWANT, M. M. – KINAGE, K. S. User Authentication Using Keystroke Latency. In *iPGCON 2014, STES's, SKNCOE*. 2014.

[44] SONG, D. X. – WAGNER, D. A. – TIAN, X. Timing analysis of keystrokes and timing attacks on ssh. In *USENIX Security Symposium*, 2001, 2001.

[45] TAHIR, M. et al. A novel network user behaviors and profile testing based on anomaly detection techniques. *International Journal of Advanced Computer Science and Applications*. 2019, 10, 6, s. 305–324.

[46] TS'O, T. – ALTMAN, J. Telnet Authentication Option. RFC 2941, RFC Editor, September 2000. Dostupné z: `https://www.rfc-editor.org/rfc/rfc2941.txt`.

[47] VINAYAK, R. User Authentification Using Advanced Keystroke Analysis. 2015.

[48] VISHWAKARMA, R. – JAIN, A. K. A Honeypot with Machine Learning based Detection Framework for defending IoT based Botnet DDoS Attacks. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, s. 1019–1024. IEEE, 2019.

Appendix A

The usage of credentials

The usage of credentials in all login attempts to the Telnet server with a public IP address in 70 days. The network traffic was captured from 17 October 2019 to 25 December 2019. The total number of logins attempts was 158 295.

Table A.1: The most used credentials to log in to the Telnet server

Username	**Password**	**Usage**	**Usage in %**
root		70 853	44.76009981
admin	1234	1 941	1.226191604
root	aquario	1 940	1.225559872
ubnt	ubnt	1 146	0.7239647494
root	root	1 052	0.6645819514
admin	admin	1 017	0.6424713352
Admin	5up	978	0.6178337913
guest	guest	978	0.6178337913
root	666666	978	0.6178337913
root	hi3518	978	0.6178337913
admin	smcadmin	977	0.6172020594
admin	vertex25ektks123	977	0.6172020594
guest	friend	977	0.6172020594
root	user	977	0.6172020594
root	vizxv	977	0.6172020594

Continued on next page

Table A.1 – *Continued from previous page*

Username	**Password**	**Usage**	**Usage in %**
root	xc3511	977	0.6172020594
tech	tech	977	0.6172020594
admin	54321	976	1.116168432
root	1234567890	976	1.116168432
root	888888	976	1.116168432
root	oelinux123	976	1.116168432
root	pass	976	1.116168432
support	support	976	1.116168432
admin	12345	975	1.115024816
admin	1234567890	975	1.115024816
guest	12345	975	1.115024816
root	1234	975	1.115024816
root	54321	975	1.115024816
root	ttnet	975	1.115024816
user	user	975	1.115024816
666666	666666	974	1.113881201
888888	888888	974	1.113881201
admin	1111	974	1.113881201
admin	7ujMko0admin	974	1.113881201
admin	ipcam_rt5350	974	1.113881201
admin	pass	974	1.113881201
admin	Win1doW$	974	1.113881201
root	00000000	974	1.113881201
root	anko	974	1.113881201
root	dreambox	974	1.113881201
root	system	974	1.113881201
root	Zte521	974	1.113881201
service	service	974	1.113881201
supervisor	zyad1234	974	1.113881201

Continued on next page

Table A.1 – *Continued from previous page*

Username	**Password**	**Usage**	**Usage in %**
admin	123456	973	1.112737586
admin	1988	973	1.112737586
admin1	password	973	1.112737586
root	1001chin	973	1.112737586
root	1111	973	1.112737586
root	123456	973	1.112737586
root	7ujMko0admin	973	1.112737586
root	alpine	973	1.112737586
root	hunt5759	973	1.112737586
admin	meinsm	972	1.111593971
Administrator	admin	972	1.111593971
root	12345	972	1.111593971
root	default	972	1.111593971
root	founder88	972	1.111593971
root	ikwb	972	1.111593971
root	juantech	972	1.111593971
root	Win1doW$	972	1.111593971
root	zsun1188	972	1.111593971
admin	admin1234	971	1.110450356
administrator	1234	971	1.110450356
mother	fucker	971	1.110450356
root	jvbzd	971	1.110450356
root	klv123	971	1.110450356
root	xmhdipc	971	1.110450356
root	zlxx.	971	1.110450356
admin	password	970	1.10930674
root	1234qwer	970	1.10930674
root	GM8182	970	1.10930674
root	password	970	1.10930674

Continued on next page

Table A.1 – *Continued from previous page*

Username	**Password**	**Usage**	**Usage in %**
supervisor	supervisor	970	1.10930674
admin	1111111	969	1.108163125
root	123123	969	1.108163125
root	5up	969	1.108163125
root	7ujMko0vizxv	969	1.108163125
root	admin	969	1.108163125
root	ivdev	968	1.10701951
root	oelinux1234	968	1.10701951
root	realtek	968	1.10701951
admin	zhongxing	967	1.105875895
default	antslq	967	1.105875895
root	klv1234	967	1.105875895
root	qazxsw	966	1.10473228
root	cat1029	924	1.056700441
admin		827	0.9457697674
admin	cat1029	46	0.02905966708

Appendix B

Guidelines for user testing

Thank you for participating in the user testing of the Telnet profiling tool. Our research focuses on profiling users in the Telnet traffic based on what and how they type. We capture and process every single packet going from your computer to our Telnet server and back. You will be given credentials to log in and set of questions which we would like you to answer.

If at any moment (during or after the experiment) you decide you don't want to participate in this testing, please let us know and we will delete all the captured data related to you.

B.1 Data we capture

- The public IP address from which you access the Telnet server. This data is used to compare where the login came from. If you want to hide your identity more, you can use a VPN or any public WiFi access point.
- Everything you type in the command line after you log in to the Telnet server. Since Telnet protocol is not encrypted, please keep in mind that we can read everything you write (even the commands you delete and don't send to the server). Because of this, don't use any personal information that may identify you - your name, username, email address, ...
- Our method is based on the characteristics of one's typing - the keystroke dynamics method. We will store the profile of how you type. This profile will be stored with a user's ID number without any identification of yourself.

B.2 Instructions

- In each session, you will be given a set of question which we ask you to answer. The order of how you answer the question is not fixed, you can start with any of them.

- Each session is very short, working on each of them should not take you longer than 10 minutes. If you know most of the commands, you will finish each session even faster.
- The answer to all questions should be marked into a log file in the Telnet server. Create and edit the log files only if your home folder. For logging information, please use the VI program. Basic commands can be found at the end of this document.
- Make sure that each set of questions and answers to them is in its own session, meaning you need to log in before you start working on the questions and log out after finishing the assignment.
- Please answer each question in sentences/longer text rather than just a number as an answer. Our goal is to learn the characteristics of you typing, so the more you type, the better.
- Do not use any of your personal information - your name, username, email or any other information. The session that you created is going to be labelled with your user ID which will be chosen randomly.
- You can make those assignments at any time of day, any day of the week. You can make all of them at once, just keep in mind that each assignment needs to be in a different session (don't forget to log out and log in after each scenario).
- Make sure you finish each assignment at least once. If you have time and you want to help us more with this research, you can go and make each assignment multiple times. The only rule here is not to work on the same assignment over and over again in a short period of time to prevent bias from learning the commands.
- Thank you if you decide to work on each assignment more than once. You can use the same log file for each attempt. You can append new information to the log file or create a new log file (please, do not delete any of log files on the server).
- During the tests, you are allowed to search for any commands that you need using any source of information (Internet, books, notes, asking someone - in this case, you should be the one who is typing the command). You are also allowed (sometimes required) to install tools you may need.
- This is not an exam. Please try to be as natural as possible while working in the command line. Please do not delete everything from the Telnet server and do not attack other devices.

B.3 Assignments

Assignment 1

- Log in to the server using given credentials.

- In the home folder create a folder named `~/noitamrofni/sresu-gig/`. In this folder, create a log file. Choose the name of the log file longer than 10 characters.
- Log in the start date and time of this assignment to the created log file.

Answer the following questions in any order you wish:

- From which IP addresses did your user logged in previously? List all of them.
- Which users has access to the Telnet server?
- When was the last time your user logged in?
- Which users are logged in to the Telnet server right now?
- What were the last 10 commands used by your user?
- For how long was your user logged in during last three logins?
- What is the usage of disk space in the device?

Assignment 2

- Log in to the server using given credentials.
- In the home folder create a folder named `~/mentlopvede/ecived-deq/`. In this folder, create a log file. Choose the name of the log file longer than 10 characters.
- Log in the start date and time of this assignment to the created log file.

Answer the following questions in any order you wish:

- Which processor is the server running on?
- What is the size of the memory available memory in the Raspberry Pi?
- Which model of the Raspberry Pi is the server running on?
- How many cores does the processor have?
- How much of free memory is available in the server?
- How many tasks are scheduled in the crontab of the device?
- When are scripts in crontab executed?

Assignment 3

- Log in to the server using given credentials.
- In the home folder create a folder named `~/krownfoinras/leewhop-ten/`. In this folder, create a log file. Choose the name of the log file longer than 10 characters.
- Log in the start date and time of this assignment to the created log file.

Answer the following questions in any order you wish:

- For how long is the server running?
- What is the IP address of the server?
- What other devices are in the internal network?
- Which ports are opened in the server?
- Which ports are opened in the rest of the devices in the internal network?
- Which interfaces are in the device? What are the MAC addresses?

Assignment 4

- Log in to the server using given credentials.
- In the home folder create a folder named `~/solving-quizzes/`. In this folder, create a log file. Choose the name of the log file longer than 10 characters.
- Log in the start date and time of this assignment to the created log file.

Answer the following questions in any order you wish:

- Which files were modified in the past 48 hours?
- How much of available disk space is in the device?
- What is the location of the `topsecret.txt` file?
- Move the `topsecret.txt` file to your home folder into the `very-secret` folder.
- Answer questions in the secret file to the log file.